THE HECKMAN BINDERY, INC.
North Manchester, Indiana

BINDING COPY

H or V	JUST	FONT	SLOT	TITLE
H	CC	1W		FACULTY WORKING 19 PAPER
H	CC	1W	8	1988
			7	NO. 1468-1485
H	CC	1W		330 B385 no.1468-1485 cop. 2
H	CC	7W		<IMPRINT> U. of ILL LIBRARY URBANA

PERIODICAL ☐ CUSTOM ☐ STANDARD ☐ ECONOMY ☐ THESIS
BOOK ☐ CUSTOM ☐ MUSIC ☐ ECONOMY AUTH 1ST ☐

ACCOUNT: 66672 | LIBRARY | NEW | RUB OR SAMPLE | TITLE I.D. | COLOR: FOIL WHT | MATERIAL

ACCOUNT NAME: UNIV OF ILLINOIS

ACCOUNT INTERNAL I.D.: E01911401 | ISSN

I.D. #2: STX4 | NOTES | BINDING FREQUENCY | WHEEL | SYS I.D.: 29276

COLLATING: 35

ADDITIONAL INSTRUCTIONS

Dept=STX4 Lot=201 Item=135 HNM=[ZY#
1CR2ST3CR MARK BY # B4 91!

SEP. SHEETS | PTS. BD. PAPER | TAPE STUBS | CLOTH EXT. | FILLER | GUM | STUB

POCKETS: PAPER | BUCK | CLOTH | SPECIAL PREP. | LEAF ATTACH

INSERT MAT. | ACCOUNT LOT NO.: 201 | JOB NO.

PRODUCT TYPE: 1 | ACCOUNT PIECE NO.: 5 | PIECE NO.: 47

HEIGHT | GROUP CARD | VOL. THIS TITLE

COVER SIZE: X

002247991

BEBR

FACULTY WORKING PAPER NO. 1483

Learning Model Management Knowledge in Intelligent Decision Support Systems

Michael J. Shaw

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign

BEBR

FACULTY WORKING PAPER NO. 1483

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

August 1988

Learning Model Management Knowledge in Intelligent Decision Support Systems

Michael J. Shaw, Assistant Professor
Department of Business Administration

LEARNING MODEL MANAGEMENT KNOWLEDGE IN INTELLIGENT DECISION SUPPORT SYSTEMS

Abstract

Model management systems are important for handling complicated decision problems in decision support systems (DSS). The current model management systems usually automate the model manipulation tasks through deductive inference mechanisms with some inherent weaknesses. Aiming at overcoming these weaknesses, we present a new framework of model management system which is able to perform model manipulation more effectively. The new approach incorporates machine learning to acquire model manipulation knowledge, stored in the form of schemata, and to refine these acquired schemata. In addition, we also address the issue of learning model selection heuristics, making the selection adaptive to the characteristics demonstrated by the problems or the users of the DSS environment.

Keywords: Model Base Management, Machine Learning, Intelligent Decision Support, Artificial Intelligence

1. Introduction

Due to the operational characteristics of decision support systems (DSSs), the solution process usually involves transforming data in various ways through a diverse collection of program modules--i.e., models. It is therefore necessary to have not only a comprehensive collection of such models (i.e., a mode bank), but also suitable mechanisms for using these models effectively. Thus, an effective model management subsystem is quite essential for solving problems and handling queries in DSSs.

This paper is aimed at applying machine learning methods to two important aspects of model management: model representation and model manipulation. Model representation concerns representing each model with its input and output conditions (Elam and Henderson [1983]; Dolk and Konsynski [1984]; Applegate et. al., [1986]; Fedorowicz and Williams [1986]). The representational approaches employed to date include predicate calculus (Bonczek et. al., [1981, 1983]), semantic network (Elam et. al., [1980]), frame (Dolk and Konsynski [1984]), and relational database theory (Blanning [1986]). All these systems basically treat models as a form of data transformation, so that the user can easily query the system without the burden of programming details, and that the model management subsystems can be easily integrated into the decision support system (Geoffrion [1987]). Using the concepts developed in machine learning, we will use schemata to represent the synthesis of multiple model applications.

Model manipulation, on the other hand, involves selecting, retrieving, and activating models to solve problems (Blanning [1986];

Dutta and Basu [1984]). That is, while individual models are used to perform stand-alone computation (e.g., time-series, simulation, regression analysis, etc.), they often need to be combined with one another into a sequence of steps in order to reach the solution. Such a process requires dynamically selecting the necessary models, imposing an appropriate sequence of model applications, and determining the desirability of each model to different decision problems. These tasks constitute model manipulation.

Prior research in model management has attempted to design model management systems capable of model manipulation in response to different problems (Bonczek et. al., [1983]; Dutta and Basu [1984]; Blanning [1986]; Dolk and Konsynski [1984]). But these systems show several weaknesses: (1) the performance of the DDS relies heavily on a predetermined collection of problem-solving methods acquired from domain experts; (2) similar problems are solved individually and independently; and (3) past problem solving experiences are ignored in solving subsequent problems. These systems ignore the fact that problem-solving skills and modeling knowledge provided by human experts may not be complete initially, and that even a commonly used solution process may change over time.

This paper presents a new framework for model management. Machine learning, an emerging technique in artificial intelligence (AI), is applied to incorporate an element of adaptiveness in the DSS. Recognized as the essential feature of any intelligent system, learning processes include the acquisition of new declarative knowledge, the development of problem-solving skills through instruction or practice,

the organization of new knowledge into general, effective representations, and the discovery of new facts and theories through observation and experimentation. Machine learning is concerned with the computer modeling of the learning processes. We will discribe the application of machine learning to model management, resulting in a learning augmented system which not only can perform problem solving intelligently, but also can accumulate prior problem-solving knowledge and refine/modify its knowledge continuously.

In addition to the generation and modification of model manipulation knowledge, the issue of using heuristics to select models adaptively will also be addressed. The method can be used to create heuristics learned from prior experiences of model selection among alternatives. We shall discuss machine learning techniques which can incrementally modify model representation by experimenting with observations; the heuristics can be intelligently created by dynamically refining an evaluation function.

The remainder of this paper is organized as follows: Section 2 presents the learning-augmented framework for intelligent DSSs, adding a learning component to the operational design of DSSs; Section 3 discusses the learning of model-manipulation schemata; Section 4 describes a learning-by-experimentation method for refining model manipulation schemata; Section 5 applies a learning method for generating model-selection heuristics adaptively; finally, Section 6 summarizes the characteristics of applying machine learning to model management.

2. A New DSS Framework Incorporating Machine Learning

Machine learning methods can be categorized into the following areas based on their behavioral characteristics: rote learning (Samuel [1968]), learning from instruction (Davis [1979]), learning by induction (Buchanan and Mitchell [1978]; Dietterich and Michalski [1983]), learning by analogy (Winston [1979]; Carbonell [1983]), learning by competition (Holland [1986]), and learning from observation and discovery (DeJong [1986]; Langley [1981]; Lenat [1983]).

A basic machine learning model is summarized in Figure 2.1, where the learning system consists of four elements: Environment, Learning Element, Knowledge Base, and Performance Element. The Learning Element takes its input from the Environment, in the form of observations, or from the Performance Element, in the form of performance results. The learning process will result in either new knowledge for the knowledge base or modifications on the existing knowledge. We shall adapt this basic model to the intelligent DSS setting, where the input from the Environment is collected from the firm's database, and the Performance Element corresponds to the rule-based problem solver of the DSS.

Insert Figure 2.1 Here

The Knowledge Base in the DSS setting contains: (1) procedural knowledge, (2) decision heuristics, and (3) model-manipulation knowledge. Procedural knowledge is the knowledge about the essential steps, mostly related to information collection, for making a given decision. The decision heuristics are rules of thumb used by domain

experts. Because of the inherently judgmental nature, this type of rules needs considerably more effort to obtain and refine. The rules generated by inductive learning belong to this category. The third type of rules is used to represent the model knowledge available for decision support; these rules indicate the application requirements of each model and the relations between models. Some examples of rules of this type are shown in Appendix A.

Most existing DSSs use knowledge engineering for knowledge acquisition; they take the domain knowledge from experi-enced decision makers in the field and transform the knowledge into the representation form in the knowledge-base of the DSS (Elain and Henderson [1980]). This is shown as process (a) in Figure 2.2. There are two types of rule learning: (1) Learning from an example set, in which decision rules are derived from a given set of positive and negative examples (shown as process (b) in Figure 2.2); and (2) Rule modification, in which the rules in the knowledge-base are modified to improve the performance of the DSS (shown in process (c) in Figure 2.2). Learning from examples can be achieved by inductive inference (Rendell [1986]; Michalski [1983]). Rule refinement, on the other hand, can be achieved by comparing the resulting solution path (i.e., the performance trace) with the correct path (i.e., the ideal trace). Bundy [1985] reviewed several methods for rule refinement and compared their performances. Learning model management knowledge involves both aspects of learning.

Insert Figure 2.2 Here

Our approach incorporates four interactive functional components--the Instance Selector, Problem-Solver, Critic, and Learning Module--to integrate the learning function. The Instance Selector either accepts the training instances supplied externally or generates new training examples by itself in response to previous learning process . The Problem-Solver produces solutions to the new problems supplied by the Instance Selector either by applying existing problem-solving heuristics or by utilizing the inference mechanism. The resulting solution path for each new problem is then evaluated by the Critic, which compares the solution just produced with the desired solution. Based on the observations made by the Critic, the Learning Module either refines existing rules or hypothesizes new rules. This learning augmented DSS configuration for knowledge refinement is shown in Figure 2.3.

Insert Figure 2.3 Here

The major applications of machine learning to model management are in three aspects: (1) the acquisition of model manipulation knowledge, (2) the refinement of model manipulation knowledge, and (3) the creation of model selection heuristics.

3. The Acquisition of Model Manipulation Knowledge

We use the term model manipulation schemata to represent the knowledge generated from past problem solving tasks. Every schema contains a condition part which describes a class of problems and a solution part which displays the shared solution plan for every

problem in this class. The solution plan can be represented in an AND/OR tree structure, which we call the solution tree. An OR subtree in the solution tree denotes all possible alternative solution paths, and an AND subtree indicates the input requirements for a model or a set of subproblems for a decomposable problem. The subproblems can be simple data retrievals or model executions. It should be noted that several models may generate similar solutions to a problem which altogether constitute an OR tree for this problem. Each subtree converging to an OR node is an alternative solution plan to this problem. Nodes in the bottom of the solution tree are either solvable terminal nodes which are subproblems solved by either data-retrieval or user input, or they may represent unsolvable terminal nodes which are subproblems that cannot be solved by data-retrieval, user input or models. The solution tree with at least one solution path whose terminal nodes are all solvable is complete since this solution tree can provide a solution plan to the problem. Otherwise, it is incomplete since it cannot provide any solution plan to the problem. The solution of a stored model manipulation schema is applicable only if a new problem matches with the condition part of this schema. The application of a model manipulation schema in the form of an AND/OR tree is shown in Figure 3.1. We use the schemata as problem solving concepts. That is, useful schemata will be those that organize operators to achieve an important goal, or a set of goals, in a general way.

As the model management system usually deals with executing two or more models in an appropriate sequence, the process of model manipulation involves a multiple-step process, in which each step involves

either a database retrieval or a model application. As opposed to searching for the individual steps, a learned model-manipulation schema integrates the entire multiple-step process into a single module; such a schema can be applied either as a single step or just a portion of it, depending on the problem to be solved (Figure 3.2).

The learning of model manipulation schemata can be characterized as "learning of multiple-step tasks," which is also used in Fikes [1972]; Korf [1982]; and DeJong [1986]. Moreover, the concept of model manipulation schemata is similar to the "macro-operators" used in (Fikes [1972] and Korf [1982]) for representing the sequence of actions learned. The macro-operators help reduce the amount of search required on the same type of problems, because they are stored in a generalized form that allows similar situations to be applied (Fikes [1972]). The learning procedure using the learning components in Figure 2.2 for acquiring model manipulation knowledge is depicted in Figure 3.3. DeJong [1986] employed "schemata" to achieve the same purpose, and he also addressed an alternative machine learning approach called explanation based learning (EBL). EBL is characterized by its use of a structured set of domain knowledge and a generalization process based on a single example (Mitchell et. al., [1986]).

Insert Figures 3.1, 3.2 and 3.3 Here

In addition to automatically acquiring model manipulation schemata, machine learning techniques also enables the model management system to refine these schemata after an iterative experimentation process. We shall elaborate on this experimentation process next.

4. Refinement of Model Manipulation Knowledge

The model manipulation schema is derived from generalizing of a specific problem instance and its solution plan. However, such a generalization may cover more than it is supposed to. To increase the accuracy of the initially learned schema, the model management system needs to modify the schema through a training process which contains a collection of self-created or teacher-provided training examples. Since the system would choose and manipulate the training instances (by the Instance Selector) in order to verify the hypothese about the concept, this process is sometimes referred to as learning by experimentation (Mitchell, Utgoff, and Banerji [1983]). The series of experiments with training instances would help the learning process converge to the correct concept description.

As defined in the preceding section, a schema consists of two parts: a condition part describing a class of problems to which this schema is applicable, and a solution part which displays the shared solution plan for every problem in this class. An experiment with a training instance provides a positive or a negative example for the current schema. A positive example has a complete instantiated solution plan based on the schema. A negative example, on the other hand, is a problem instance which does not belong to the class under consideration. After the model manipulation schema is acquired by generalizing the derived solution plans for the given problem, the refinement of the schema on the current over-generalized form is achieved by an iterative process generalizing or constraining the training examples. This refinement process can be summarized by the

following two operations: If the current problem expression of the schema does not cover the encountered positive example, then it needs to be generalized. If the current problem description of the schema covers the encountered negative example, then it needs to be constrained. This refinement process can be facilitated by organizing the possible problem descriptions in a "version space" (Mitchell [1982]).

Essentially, we are treating the refinement process as a search process for concept learning--in this case, the concept to be learned is the correct problem description for the schema. The search is conducted in the space of all possible versions of the descriptions, referred as the version space. The version space basically provides a generality/specificity structure for guiding the refinement process. Given a version space and a description in the version space, the Learning Module should be able to find the more generalized version of the description, the more specific version of the description, or descriptions belonging to the same level of generalization.

The approach described in Mitchell [1982] takes advantages of the general-to-specific ordering of descriptions in the version space. Mitchell argues that a version space can be represented by two sets of descriptions, S and G, where S is the set of the most specific descriptions consistent with the observed instances, and G is the most general descriptions consistent with the observed instances.

To refine a model manipulation schema, we first create a version space for the problem descriptions to which this schema is applicable. This version space is represented by the G and S sets. Initially, G

and S are defined by the first training example t_o: G is the maximal generalization of t_o and S is defined to be t_o. The set of training examples are then input sequentially to shrink the version space. For each example, if the training example is a positive instance, then (1) generalize S, as little as possible, in order to cover this positive example and (2) remove from G all concept descriptions that do not cover the example; on the other hand, if the training example is negative, then (1) remove from S the parts which cover the example and (2) make G more specific, as little as possible, so that its elements would not cover the example. Thus, in each step G is constrained to avoid covering the negative examples and S is expanded to cover the positive examples encountered. G and S will eventually be equal as more and more training examples are considered. When they finally converge, the proper problem description for the schema is found. The learning procedure for the refinement of model manipulation knowledge is depicted in Figure 4.1.

Insert Figure 4.1 Here

As shown in Figure 4.1, it is sometimes necessary for the Instance Selector to generate training examples to expedite the schema refinement process. The main idea is to make some slight changes on a prior training example and see if the changes would result in a different classification for the new example. By considering the new training example in the Learning Module, the S and G sets would move closer to each other. This converging process between S and G can be further facilitated if the new training examples selected represent concepts

closely related to some prior training examples. Mitchell et al. [1983] has a more detailed account on how the new training examples can be generated by the Instance Selector to facilitate the learning process.

In many learning situations, it is possible to have "near miss examples"--i.e., the instances which are very close to being positive (Winston [1979]). In refining model manipulation schemata, for example, the near miss examples can be defined as the problem descriptions which, although not directly solvable by the schema, need only minor modifications for the schema to be applicable (e.g., one unsolvable node in the solution tree when the schema is applied). The Learning Module can decide to modify the schema by finding the solution for the unsolvable node, so that it becomes applicable to this near miss example. The G and S sets are updated by treating the near miss example as a positive instance for the modified schema. The refinement process would continue until G and S converge. An example of schema refinement using positive, negative, and near-miss examples is described in Appendix B.

5. Learning Model Selection Heuristics

When there are more than one way to solve a given problem (e.g., models such as regression, moving average, exponential smoothing, and delphi models can all solve a forecasting problem), the model management system usually either let the user select the best model, or it can choose among these alternative models based on a heuristic function. This heuristic function is chosen based on past performances or

human experts' experiences and is usually in the form of a polynomial of several important factors: $E = \sum_i w_i * f_i$, where w_i is the weight given to f_i. For example, the f_is' used in a heuristic function for scoring the performances of several forecasting models could be the accuracy/error, the operating cost, the operating time, and the difficulty of collecting data for each model, where each f_i is characterized in a numeric scale.

The coefficients of the heuristic function may be affected by the preference of the users and by the characteristics of the problems. A marketing manager may think that the past accuracy of a forecasting model should dominate other criteria, but an MIS manager may give higher preference to the computational efficiency. Therefore, different users may assign different heuristic functions to the models. Hence, the model management system should be able to adjust the coefficients of the heuristic function according to the "preference patterns" manifested by the users. To that end, an inductive learning method is needed to derive the heuristic function for each user, based on observations of that user's selection behavior. Since the objective is to make model selection adaptive to the user's preference, we adopted the inductive learning method articulated in Rendell's probabilistic learning system (PLS) (Rendall [1983, 1986]). The heuristic function in this learning method corresponds to the utility function defined on a feature space. The feature space consists of a set of rectangular regions, each of which contains instances of a single concept (i.e., class). Thus, the region R in the feature space can be defined as R = (r, u, e), where r is a rectangular region in the

feature space; u is utility function value, as estimated by the probability given by the ratio of the positive instances to the total observed instances in this region, and e is the error rate allowed in this region. The utility indicates the probability that an instance in the region is a positive instance; e is used to represent the system's confidence in its judgement of the instances contained inside a region.

This PLS framework can be applied to generate the heuristics for model selection. For example, suppose that the models are ranked on two performance criteria: (1) quality of solution and (2) computational complexity, which are treated as the two dimensions of the feature space. Each region in the feature space then contains those model instances on the same utility level, described by the combination of solution quality and time complexity. For a given model, the corresponding utility—which represents the value for the model selection heuristics--can be determined by mapping it into the feature space. This approach progressively refines the utility assigned to each region by splitting a region into smaller regions. In addition, unlike some of the other learning system (e.g., Michalski [1983]), this learning method can effectively handle noisy data in the training set.

For example, in Figure 5.1a, the three problems--"the sales next year," "the inventory three years later," and "the interest expense next year"--all face the decision of choosing the best forecasting model. For the sake of simplicity, we use the quality of solution and computational complexity as two criteria for evaluating alternative

models. In the set of training examples, the chosen model for each problem is treated as a positive instance, and the rest are treated as negative instances.

Insert Figure 5.1 Here

Using each model's solution quality and computional complexity, the Learning Module can localize the models in the feature space (Figure 5.1b). To determine the heuristic value, the Learning Module further divides the feature space into several classes using the following procedure. Initially, it arbitrarily splits the feature space into two regions, and calculates the success probability, u, in each of these regions. In figure 11.b, an arbitrary splitting generates the regions, r_1 and r_2, which initially have the utilities (probabilities) $u_1 = 2/7$, and $u_2 = 1/5$, respectively; they are estimated by the ratio of positive instances to total instances in each region. Each region is then refined by a further splitting, where the best splitting is the one resulting in the largest dissimilarity d among all possible partitions of the region. Rendell defined the dissimilarity measure, d, for each splitting as $(|\log u_1 - \log u_2| - \log(e_1/e_2))$, where u_1, u_2, and e_1, e_2, are the utilities and error rates for the two regions after the splitting. This splitting process is repeated until $d \leq 0$ for every region, shown as Figure 4.5c in the example. Every region can then define a utility class, in which the models are of the same preference level. This inductive learning process can be applied to the training examples collected from the individual users; the utility classification derived from a set of

training examples reflects the preference of that user and can be used as the heuristic for model selection.

6. Conclusions

In this paper, we have presented a learning augmented approach to the design of model base management sybsystem of DSSs by adding a Learning and Knowledge Acquisition Unit. The Learning and Knowledge Acquisition Unit can acquire decision rules through an inductive learning engine; it can also refine the rules or derive decision schemata by four functional components: the Instance Selector, the Problem Solver, the Critic, and the Learning Module. This learning augmented methodology provides a unified framework for supporting such important model management operations as rule learning and refinement, improving model manipulation, and deriving heuristics for model selection.

References

Applegate, L. M., Konsynski, B. R., and Nunamaker, J. F., 1986, "Model Management Systems: Design for Decision Support," Decision Support Systems, Vol. 2, pp. 81-91.

Blanning, R., 1986, "An Entity-Relationship Approach to Model Management," Decision Support Systems, Vol. 2, No. 1, pp. 65-72.

Bonczek, R. H., C. W. Holsapple, and A. B. Whinston, 1980, "Future Directions for Developing Decision Support Systems," Decision Science, Vol. 11, pp. 616-631.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B., 1981, "Representing Modeling Knowledge with First Order Predicate Calculus," Operations Research.

Bonczek, R. H., Holsapple, C. W. and Whinston, A. B., 1983, "Specification of Modeling and Knowledge in Decision Support Systems," in H. G. Sol (ed.), Processes and Tools for Decision Support, (North Holland: Amsterdam).

Buchanan, G. B. and Mitchell, T. M., 1978, "Model-Directed Learning of Production Rules," Pattern-Directed Inference Systems, in Waterman, D., and Hayes-Roth, F. (eds.) (New York: Academic Press).

Bundy, A., Silver, B., and Plummer, D., 1985, "An Analytical Comparison of Some Rule-Learning Programs," Artificial Intelligence, 27, pp. 137-181.

Carbonell, 1983, "Learning by Analogy: Formulating and Generalizing Plans from Past Experiences," in Machine Learning, ed. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga.

Davis, R., 1979, "Interactive Transfer of Expertise: Acquisition of New Inference Rules," Artificial Intelligence, Vol. 12, pp. 121-57.

DeJong, G., 1986, "An Approach to Learning from Observation," in Machine Learning (Vol. II), Michalski et al. (Eds.), (Morgan Kaufmann, Los Altos, CA).

Dietterich, T. G. and Michalski, R. S., 1983, "A Comparative Review of Selected Methods for Learning from Examples," in Machine Learning: An AI Approach, ed. R S. Michalski, J. G. Carbonell, and T. M. Mitchell, Tioga.

Dolk, D. and Konsynski, B., 1984, "Knowledge Representation for Model Management Systems," IEEE Trans. on Software Engineering, Vol. SE-10, No. 4, pp. 619-627.

Dutta, A. and Basu, A., 1984, "An Artificial Intelligence Approach to Model Management in Decision Support Systems," IEEE Computer, Vol. 17, No. 9, pp. 89-98.

Elam, J. J. and Henderson, J. C., 1980, "Knowledge Engineering Concepts for Decision Support System Design and Implementation," Proceedings of Fourteenth Annual Hawaii International Conference on System Sciences, (Western Periodicals: North Hollywood, CA).

Elam, J. and Konsynski, B., 1987, "Using Artificial Intelligence Techniques to Enhance the Capabilities of model Management Systems," Decision Sciences, Vol. 18, pp. 487-502.

Fedorowicz, J. and Williams, G. B., 1986, "Representing Modeling Knowledge in an Intelligent Decision Suport System," Decision Support Systems 2, pp. 3-14.

Fikes, R., Hart, P., and Nilsson, N., 1972, "Learning and Executing Generalized Robot Plans," Artificial Intelligence, Vol. 3, No. 4, pp. 251-288.

Geoffrion, A. M., 1987, "An Introduction to Structured Modeling," Management Science, Vol. 33, No. 5, pp. 547-589.

Henderson, J., 1987, "Finding Synergy Between Decision Support Systems and Expert Systems Research," Decision Sciences, Vol. 18, pp. 333-349.

Holland, J., 1986, "Escaping Brittleness: The Possibility of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," Machine Learning: An AI Approach, Michalski, Carbonell, and Mitchell (Eds.), Tioga Pub. Co., Palo Alto, CA.

Konsynski, B. and Sprague, R. H., Jr., 1986, "Future Research Directions in Model Management," Decision Support Systems, 2, pp. 103-109.

Korf, R., 1985, Learning to Solve Problems by Searching for Macro-operators, (Pitman, Marshfield, Mass.).

Langley, P., 1981, "Data-Driven Discovery of Physical Laws," Cognitive Science, Vol. 5, pp. 31-54.

Langley, P., 1984, "Learning to Search: From Weak Methods to Domain-Specific Heuristics," CMU-R1-TR-84-21, (The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA), also appeared in Cognitive Science (1986).

Lenat, D. B., 1983, "EURISKO: A Program that Learns New Heuristics and Domain Concepts; The Nature of Heuristics III: Program Design and Results," Artificial Intelligence, Vol. 21, pp. 61-98.

Michalski, R. S., 1980, "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 2, pp. 249-361.

Michalski, R. S., 1983, "A Theory and Methodology of Inductive Learning," in Michalski, R., Carbonell, J., and Mitchell, T., (eds.), Machine Learning, Tioga Publishing Co., Palo Alto, CA.

Mitchell, T., 1982, "Generalization As Search," Artificial Intelligence, Vol. 18, pp. 203-226.

Mitchell, T., Utgoff, P. E. and Banerji, R., 1983, "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristic," in Machine Learning: An Artifical Intelligence Approach, R S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), Tioga Publishing Co., Palo Alto, CA.

Mitchell, T., Keller, R., and Kedar-Cabelli, S., 1986, "Explanation-based Generalization: A Unifying View," Machine Learning, 1, pp. 47-80.

Rendell, L., 1983, "A New Basis for State-Space Learning Systems and a Successful Implementation," Artificial Intelligence, Vol. 20, pp. 369-92.

Rendell, L., 1986, "A General Framework for Induction and a Study of Selective Induction," Machine Learning, Vol. 1, No. 2, pp. 177-226.

Samuel, A. L., 1967, "Some Studies in Machine Learning Using the Game of Checkers II--Recent Progress," IBM J. of Research and Development, Vol. 11, No. 6, pp. 609-617.

Sprague, R. H. and Carson, E. D. 1982, Building Effective Decision Support Systems, (Prentice-Hall Inc.: Englewood Cliffs, NJ).

Stohr, E. A., 1986, "Decision Support and Knowledge-based System: A Special Issue," Journal of Management Information Systems, Vol. II, No. 4.

Winston, P., 1979, "Learning and Reasoning By Analogy," Communications of ACM, Vol. 23, No. 12, pp. 689-703.

Appendix A

Production Rules for Applying Models in Loan Evaluation

In a DSS, production rules can be used to represent model knowledge. The application of each model is directed by an if-then rule and interpreted as "if the input requirements are satisfied and the model thus becomes executable, then the ouput value is ..." In the model predicates, we use the upper case to specify the model, underlines to represent the input values, and the rest to represent the output values. Some of the rules directing model applications in a loan-evaluation DSS are listed here. Machine learning techniques can be used to learn additional rules or to refine existing rules.

1 (var1, ?x1, yr1, fn)&(var2, ?x2, yr1, fn)&(REGRESS, ?x1, ?x2, ?x3, ?x4, yr, fn)
= > (ß, var1, var2, ?x3, yr fn)&(R^2, var1, var2, ?x4, yr, fn)&(var1, ?x3, yr2, fn)

With the input values, ?x1 and ?x2, of var1 and var2 in a given year for a particular firm, the REGRESS model ouputs values, ?x3 and ?x4, of ß and R^2 between the two input variables.

2 (var1, ?x1, yr, fn)&(var2, ?x2, yr, fn)&(RATIO, ?x1, ?x2, ?x3, yr, fn)

= > (ratio, var1, var2, ?x3, yr, fn)

Using the input values, ?x1 and ?x2, of var1 and var2 in a given year for a given firm, the Ratio model calculates the value of their ratio, ?x3.

3 (var, ?x1, yr, fn)&(var, ?x2, (- yr 1), fn)&(var, ?x3, (- yr 2), fn)&(AVG, ?x1, ?x2, ?x3, ?x4, yr, fn)

= > (avg, var, ?x4, yr, fn)

Using the input values, ?x1, ?x2, and ?x3, of var from three consequentive years, the AVGERAGE model calculates their average value, ?x4.

4 (var, ?x1, yr, fn)&(industry-type, ?x2, yr, fn)&(PERCENTILE, ?x1, ?x2, ?x3, yr, fn)

= > (percentile, var, ?x3, yr, fn, ?x2)

Using the value of var and the industry type of this firm, the PERCENTILE model calculates its percentile value of var in its industry.

5 (var, ?x1, yr, fn)&(industry-type, ?x2, yr, fn)&(MEDIAN, ?x1, ?x2, ?x3, yr, fn)

= > (median, var, ?x3, yr, fn, ?x2)

Using the value of var and the industry type of this firm, the MEDIAN model calculates its median value of var in its industry.

6 (var, ?x1, yr, fn)&(tax-type, ?x2, yr, fn)&(TAX, ?x1, ?x2, ?x3, yr, fn)

= > (after-tax, var, ?x3, yr, fn)

Using the value of var and the tax-type of this firm, the TAX model calculates the after-tax value of var.

7 (var, ?x1, yr, fn)&(var, ?x2, yr, (-, yr, 1), fn)&(var, x3, (-, yr, 2), fn)&(TREND, ?x1, ?x2, ?x3, ?x4, yr, fn)

= > (trend, var, ?x4, yr, fn)

Using the value of var from three consequentively years, the TREND model calculates the trend of var.

8 (ratio, (+, long-term-debt, curr-liab, ?x1, yr, fn), total-assets, ?x2, yr, fn)&(ratio, funds-from-op, (+, interest, (avg, debt-maturity, ?x4, yr, fn)&(trend, sales, ?x5, yr, fn)&(RISK-SCORE, ?x1, ?x2, ?x3, ?x4, ?x5, ?x6, yr, fn)

= > (risk-score, ?x6, yr, fn)

Using (long-term-debt + current-liabilities) to total-assets ratio, and funds-from-operation to (interest + the-average-debt-maturity) ratio, the RISK-SCORE model calculates the risk score of this firm.

9 (interest-income,?x1, yr, fn)&(cost-of-handling-deposit, ?x2, yr, fn)& (avg,loan-volume, ?x3, yr, fn)&(avg, collected-balance, ?x4, yr, fn)&(risk-score,?x5, yr, fn)&(LT, ?x5, 0)&(LOAN-YIELD-I, ?x1, ?x2, ?x3, ?x4, ?x5, ?x6, yr, fn)

= > (loan-yield, ?x6, yr, fn)

Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is less than 0, the LOAN-YIELD-I model calculates the loan-yield of this firm.

10 (interest-income, ?x1, yr, fn)&(cost-of-handling-deposit, ?x2, yr,fn)& (avg,

loan-volume, ?x3, yr, fn)&(avg, collected-balance,?x4, yr, fn)&(risk-score, ?x5, yr, fn)&(GT, ?x5, 0)&(LOAN-YIELD-II, ?x1,?x2, ?x3, ?x4, ?x5, ?x6, yr, fn)

= > (loan-yield, ?x6, yr, fn)

Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 0, the LOAN-YIELD-II model calculates the loan-yield of this firm.

11 (interest-income, ?x1, yr, fn)&(cost-of-handling-deposit, ?x2, yr,fn)&(avg, loan-volume, ?x3, yr, fn)&(avg, collected-balance, ?x4, yr, fn)&(risk-score, ?x5, yr, fn)&(GT, ?x5, 1.255)&(LOAN-YIELD-III, ?x1, ?x2, ?x3, ?x4, ?x5, ?x6, yr, fn)

= > (loan-yield, ?x6, yr, fn)

Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 1.255, the LOAN-YIELD-III model calculates the loan-yield of this firm.

12 (interest-income, ?x1, yr, fn)&(cost-of-handling-deposit, ?x2, yr,fn)&(avg, loan-volume, ?x3, yr, fn)&(avg, collected-balance, ?x4, yr, fn)&(risk-score, ?x5, yr, fn)&(GT, ?x5, 2.79)&(LOAN-YIELD-IV, ?x1, ?x2, ?x3, ?x4, ?x5, ?x6, yr, fn)

= > (loan-yield, ?x6, yr, fn)

Using the interest-income, cost-of-handling-deposit, three year average loan-volume and collected-balance, and the risk-score, under the condition that the risk score is greater than 2.79, the LOAN-YIELD-IV model calculates the loan-yield of this firm.

13 (trend, interest-rate, ?x1, yr, fn)&(interest-rate, ?x2, yr, fn)&(loan-period, ?x3, yr, fn)&(BT, ?x3, 3, 12)&(ST-LOAN-RATE, ?x1, ?x2, ?x3, ?x4, yr, fn)

= > (st-loan-rate, ?x4, yr, fn)

Using the trend of interest-rate, interest-rate, and the loan-period, under the condition that the loan-period is between 3 to 12 months, the ST-LOAN-RATE model calculates the short term loan rate of this firm.

14 (trend, interest-rate, ?x1, yr, fn)&(interest-rate, ?x2, yr, fn)&(loan-period, ?x3, yr, fn)&(GT, ?x3, 12)&(LT-LOAN-RATE, ?x1, ?x2, ?x3, ?x4, yr, fn)

= > (lt-loan-rate, ?x4, yr, fn)

Using the trend of interest-rate, interest-rate, and the loan-period, under the condition that the loan-period is greateer than 12 months, the LT-LOAN-RATE model calculates the long term loan rate of this firm.

15 (interest-cost, ?x1, yr, fn)&(operating-cost, ?x2, yr, fn)&

(avg-assets, ?x3, yr, fn)&(reserve-requirements, ?x4, yr, fn)&

(COST-OF-FUNDS, ?x1, ?x2, ?x3, ?x4, ?x5, yr, fn)

= > (cost-of-funds, ?x5, yr, fn)

Using the interest-cost, operating-cost, three year average assets, and the reserve-requirements, the COST-OF-FUND model calculates the cost-of-fund of this firm.

16 (cost-of-funds, ?x1, yr, fn)&(loan-yield, ?x2, yr, fn)&

(COMPENSATING-BALANCE, ?x1, ?x2, ?x3, yr, fn)

= > (compensating-balance, ?x3, yr, fn)

Using the cost-of-funds, and the loan-yield, the COMPENSATING-BALANCE model calculates the compensating-balance of this firm.

Appendix B An Example Illustrating the Schema Refinement Process

This appendix describes an example applying the learning method described in Section 4 to refine an existing model manipulation schema. The initial schema is shown in Figure A-1, with G and S defined for the version space. The generalization relations between the domain variables are organized into a hierarchy shown in Figure A-2.

In Figure A-1, a model manipulation schema is created from an initial positive instance, (percentile, (ratio, A/R, inv), ?x1, 1986, ABC), which represents the computation modules for getting the percentile value of the ratio between accounts-receivable (A/R) and inventory (inv) in a given year (1986) for a particular firm (ABC). This initial schema has a version space where the G set is the maximally generalization of this instance, (percentile, (ratio, var1, var2), ?x1, yr, fn), base on the generalization hierarchy shown in Figure A-2. The S set is initiated to be the training instance. In Figure A-3, the schema is applied to a new instance: (percentile (ratio, asset, liab), ?x1, 1986, ABC). Since the instantiated solution tree is complete, the instance is classified as a positive example. It modifies the current version space by minimally generalizing the S set. Based on Figure A-2, asset is the minimal generalization of asset and accounts-receivable, and B/S-var is the minimal generalization of liability and inventory. Therefore, minimally generalizing S would result in, (percentile, (ratio, asset, B/S-var), ?x1, 1986, ABC).

In Figure A-4, the training instance, (percentile, (ratio, profits, assets), ?x1, 1988), has an incomplete solution tree. Consequently, this instance is classified as a negative example for the current schema. It then modifies the current version space by constraining the G set to be (percentile, (ratio, B/S-var, var2), yr, fn), (percentile, (ratio, var1, I/S-var), yr, fn), or (percentile, (ratio, var1, var2), yr, fn) (yr < 1988).

In Figure A-5, a near-miss example modifies the schema by adding one more precondition of the RATIO model, (avg, var1, ?x5, yr, fn). The G and S sets in the current version space are also updated to include the maximal and minimal generalizations of this example.

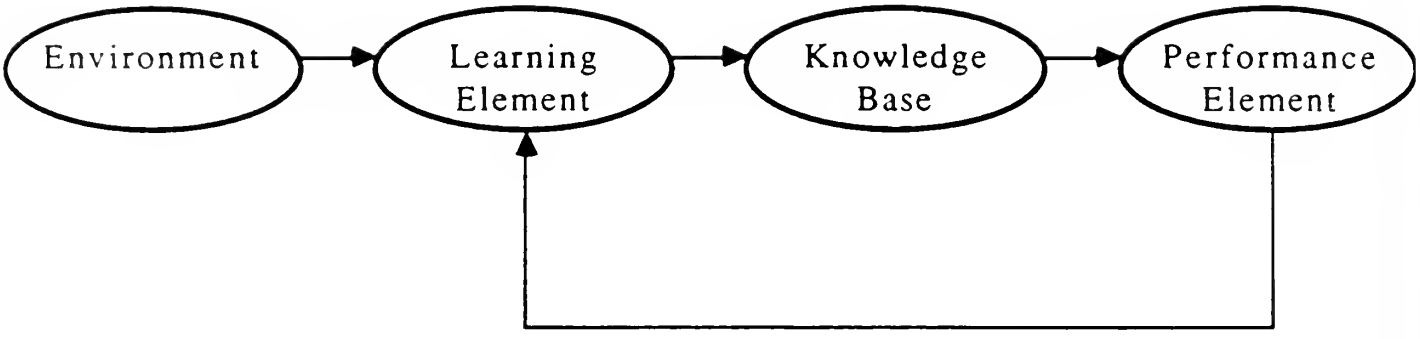


Figure 2.1 The Basic Model of a Machine Learning System

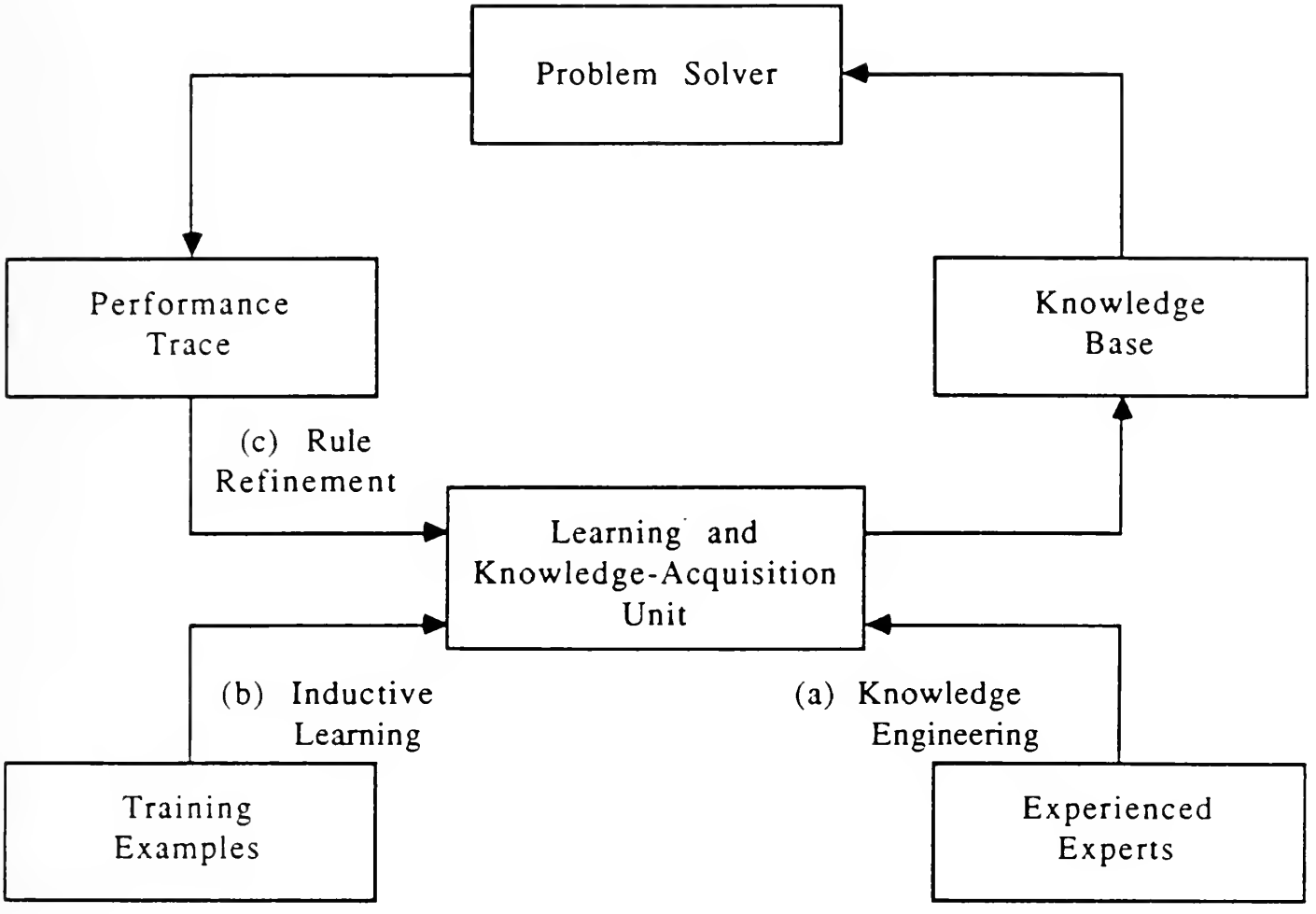


Figure 2.2 Interactions Between the Learning Module and Other DSS Components

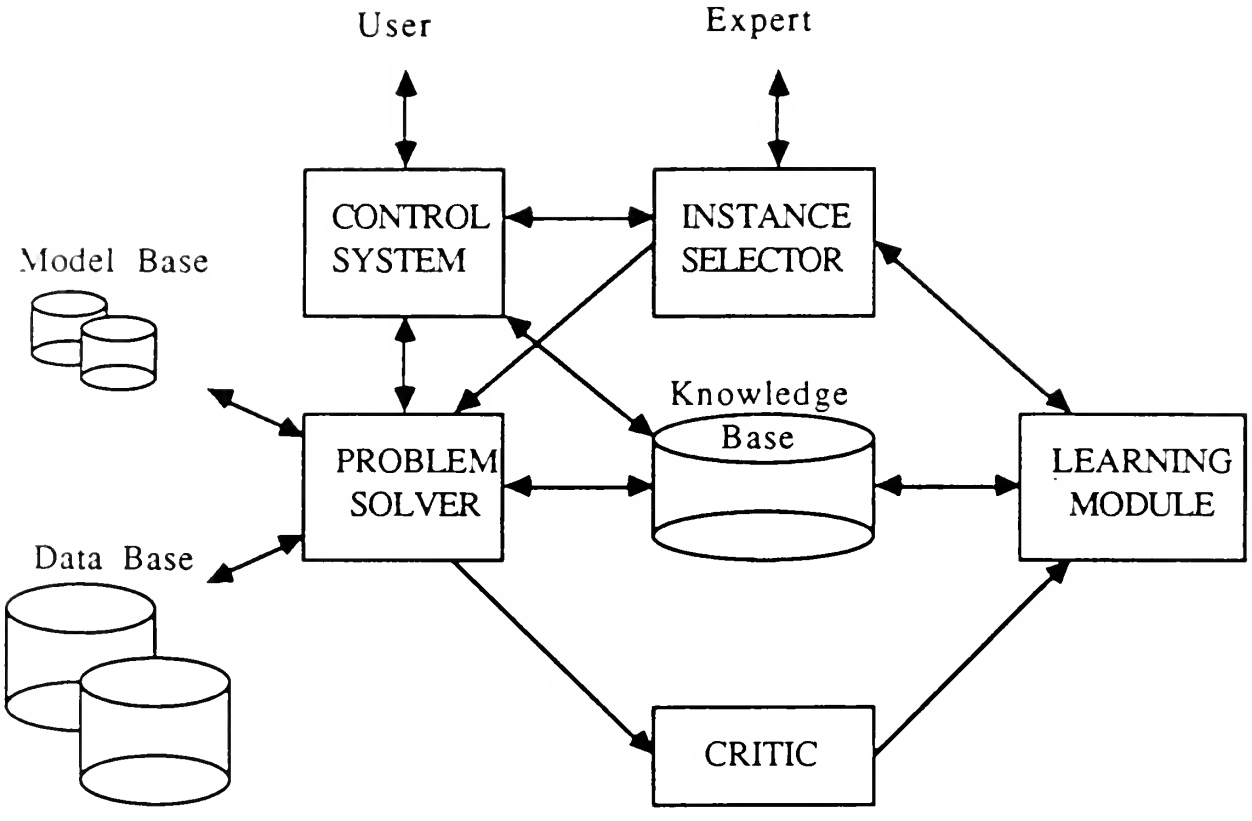


Figure 2.3 The Learning-Augmented DSS Framework for Knowledge Refinement

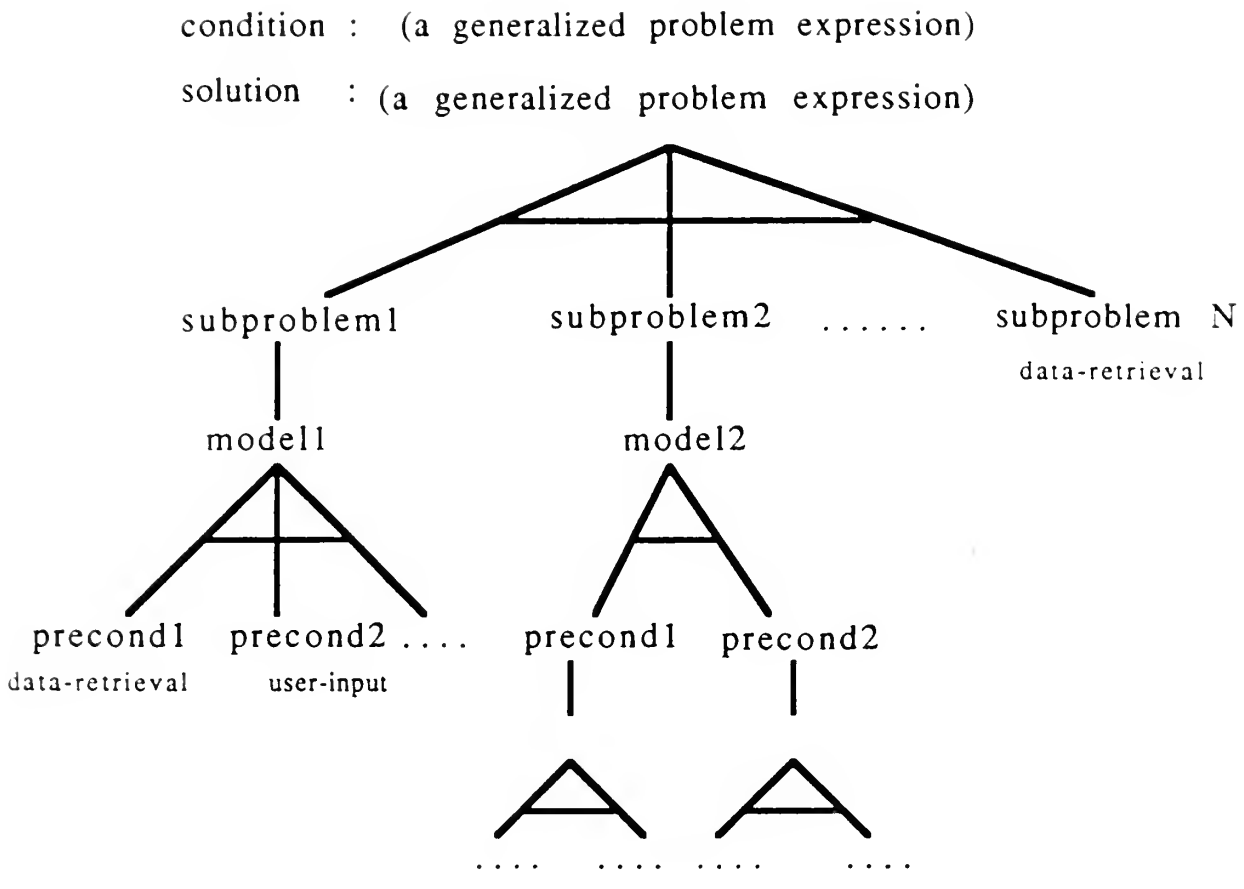
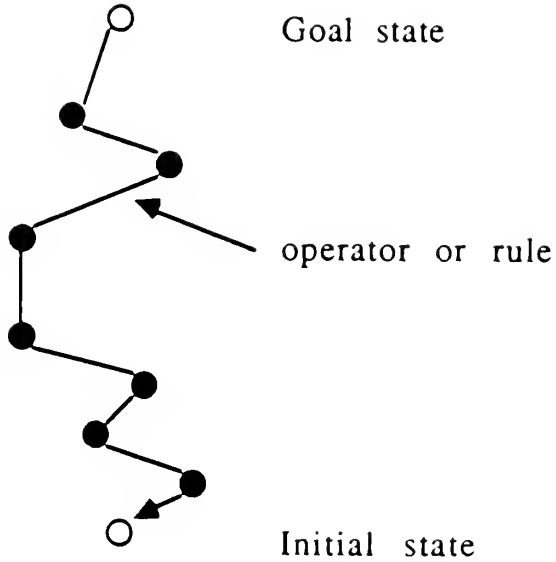
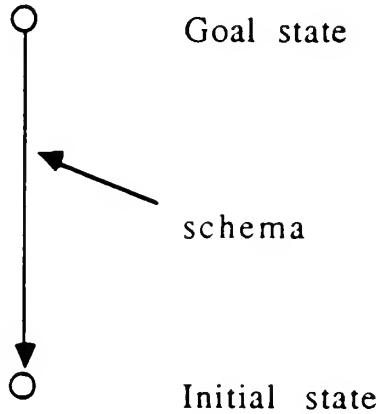


Figure 3.1 The Application of A Model Manipulation Schema



(a)



(b)

Figure 3.2 Search Processes (a) Without and (b) With Schemata

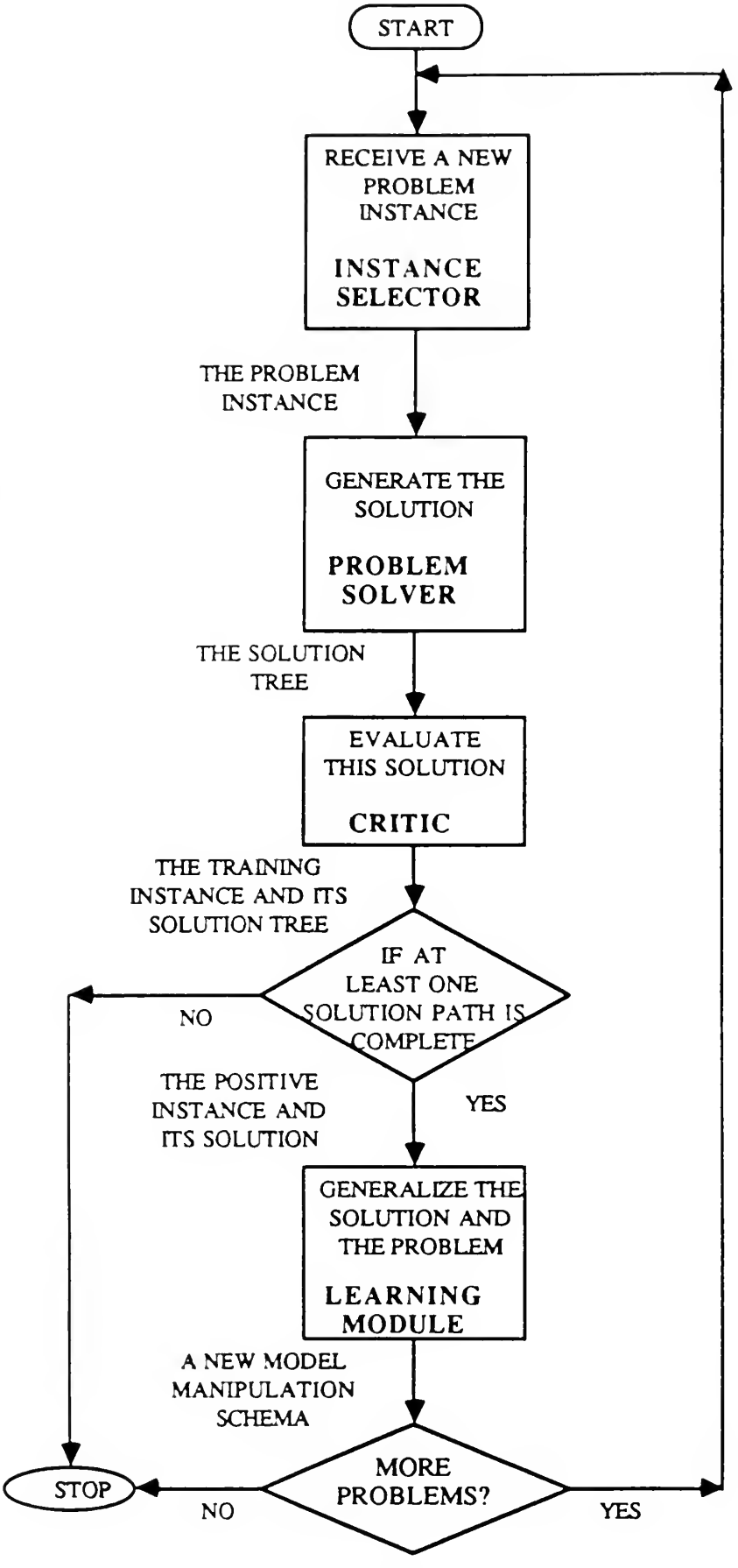


Figure 3.3 The Learning Procedure in the Acquisition of Model Manipulation Knowledge

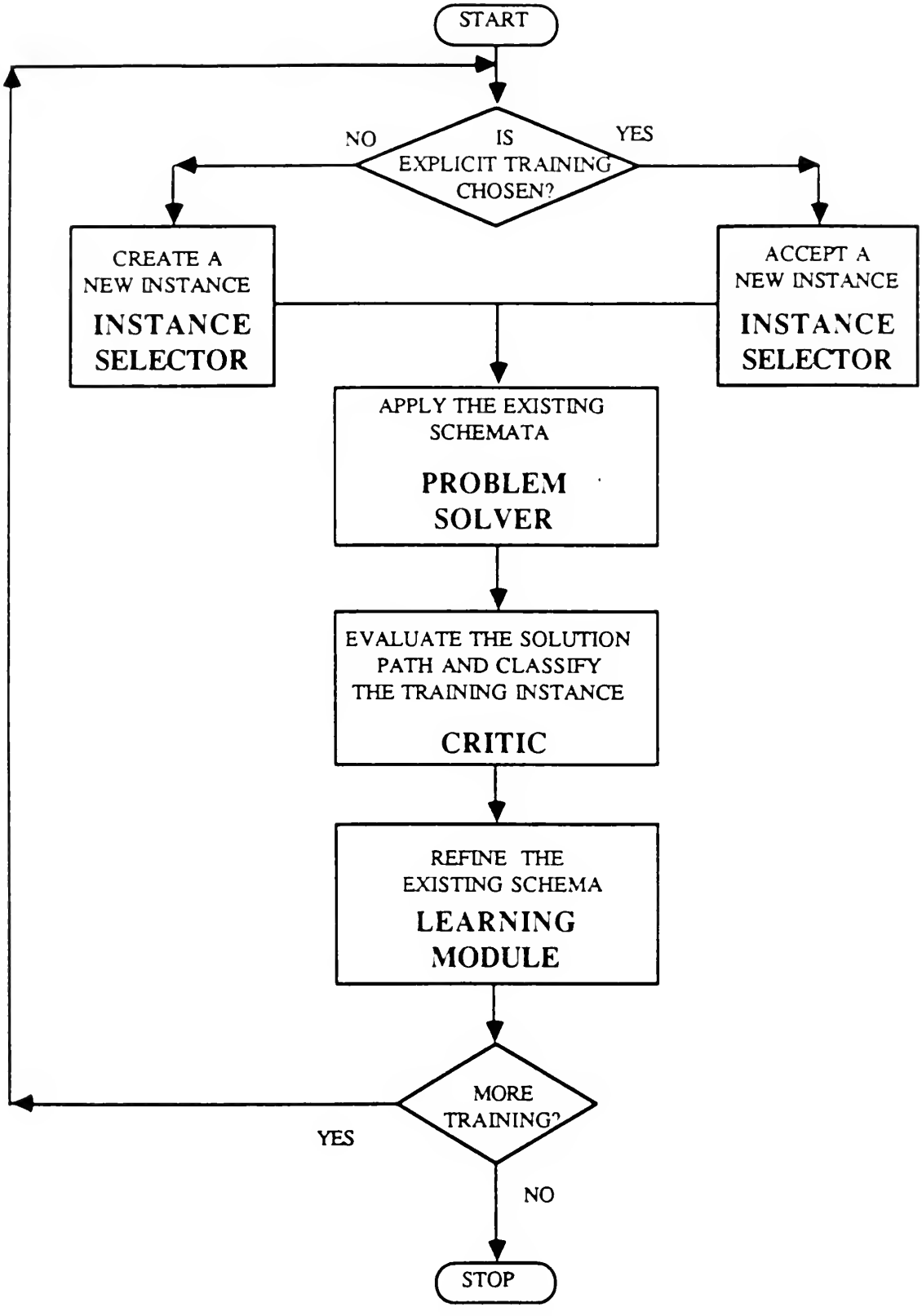
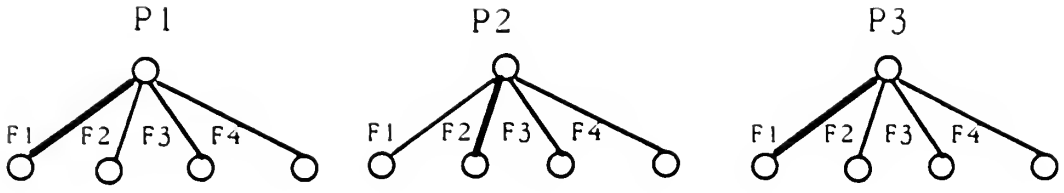


Figure 4.1 The Learning Procedure in the Refinement of Model Manipulation Knowledge

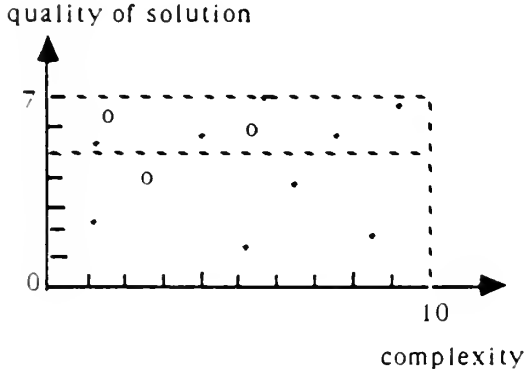
(a)



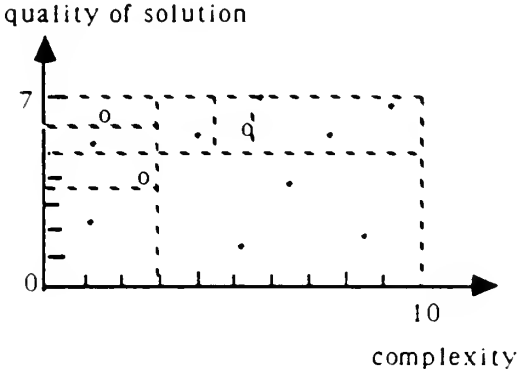
F1: REGRESSION
F2: MOVING AVERAGE
F3: EXPONENTIAL SMOOTHING
F4: DELPHI

P1: THE SALES NEXT YEAR
P2: THE INVENTORY THREE YEARS LATE
P3: THE INTEREST EXPENSE NEXT YEAR

(b)



(c)



o -- denotes a positive instance
• -- denotes a negative instance

Figure 5.1 An Example of Learning Model Selection Heuristics Using the PLS Approach

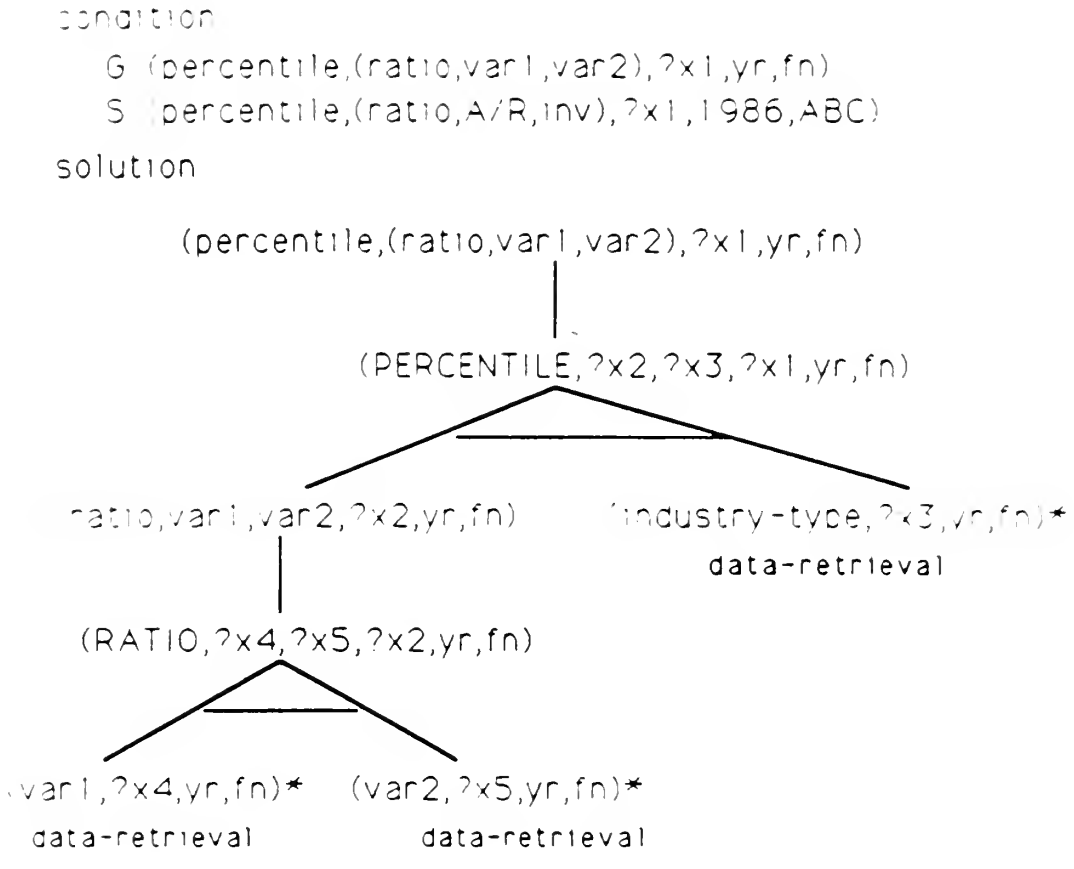


Figure A-1. The Initial Schema

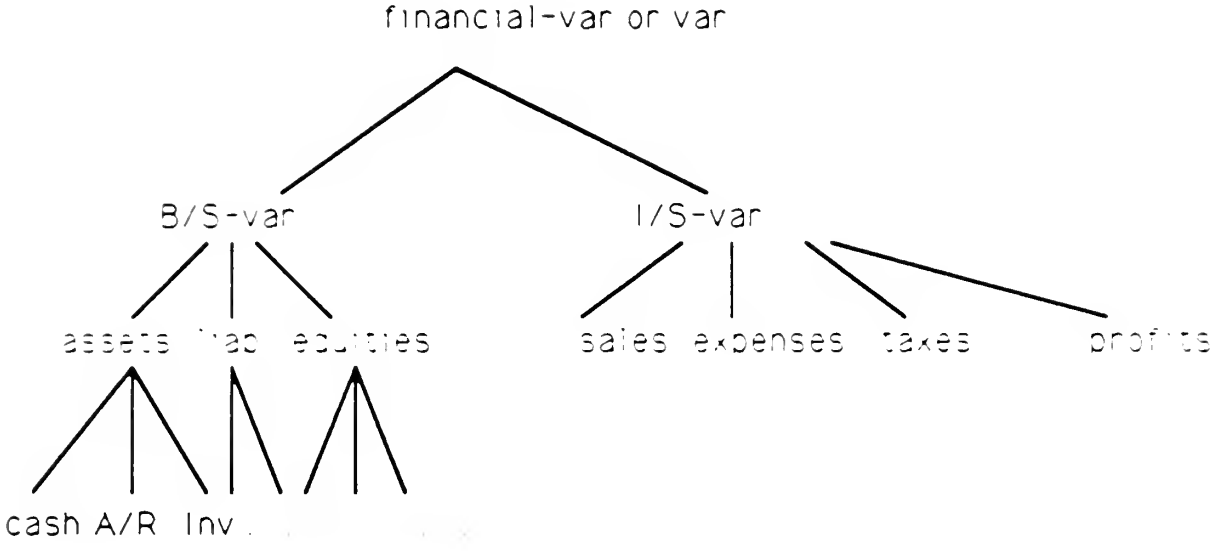
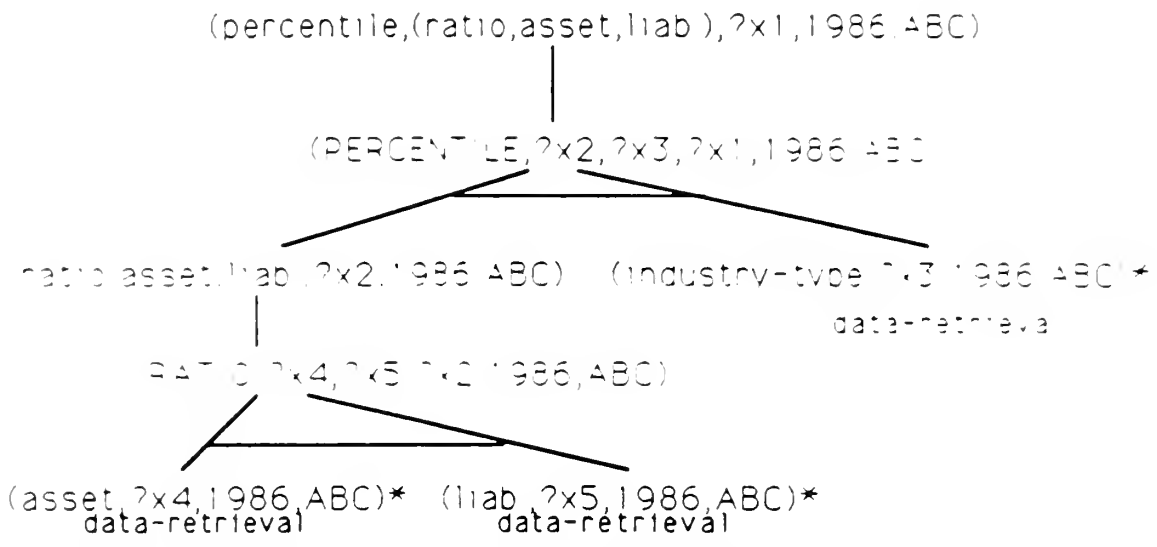


Figure A-2. The Generalization Hierarchy

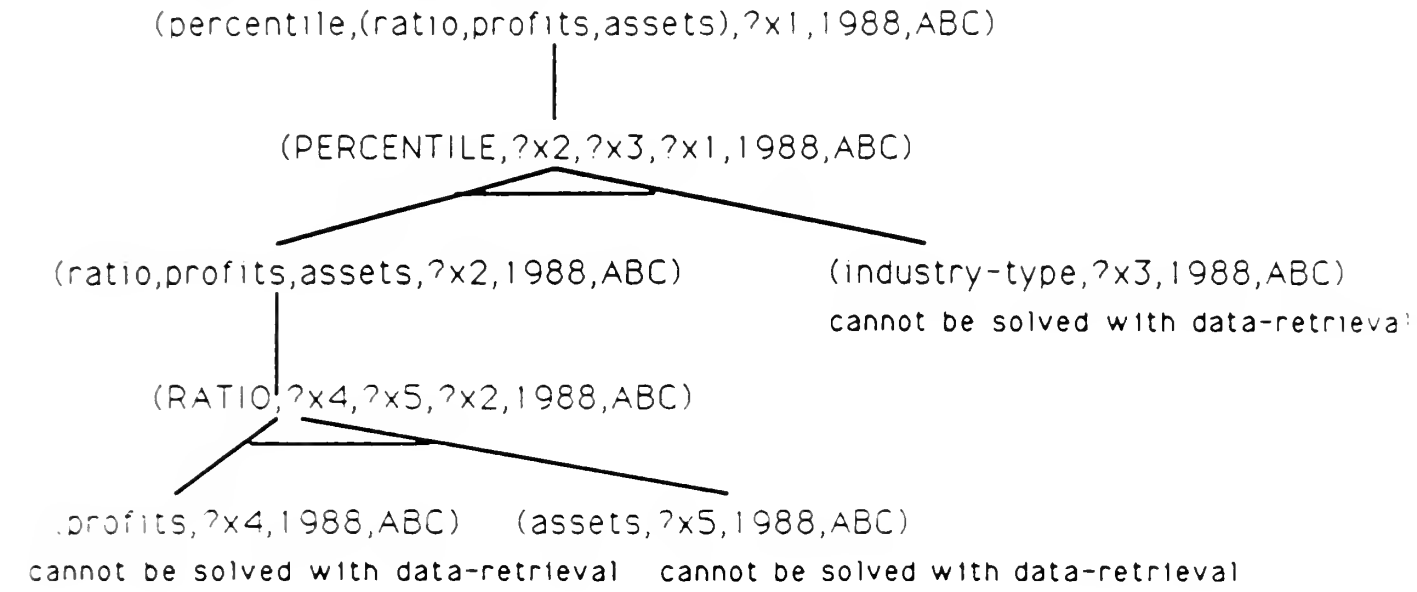
- An example,(percentile,(ratio asset,liab),?x1,1986,ABC) with the pattern-matching, (assets/var1,liab/var2,1986/yr,ABC fn), has the following instantiated solution



- This example is classified as a positive example, since all terminal nodes are solvable, It modifies the condition of this schema as follows
  - S (percentile,(ratio,var1,var2),?x1,yr,fn)
  - S (percentile,(ratio,asset B/S-var),?x1,1986,ABC)

Figure A-3. Applying the Schema to a Positive Example

- An example, (percenitle,(ratio,profits,assets),?x1,1988,ABC), with the pattern-matching, (profits/var1,assets/var2,1988/yr,ABC/fn) has the following instantiated solution:

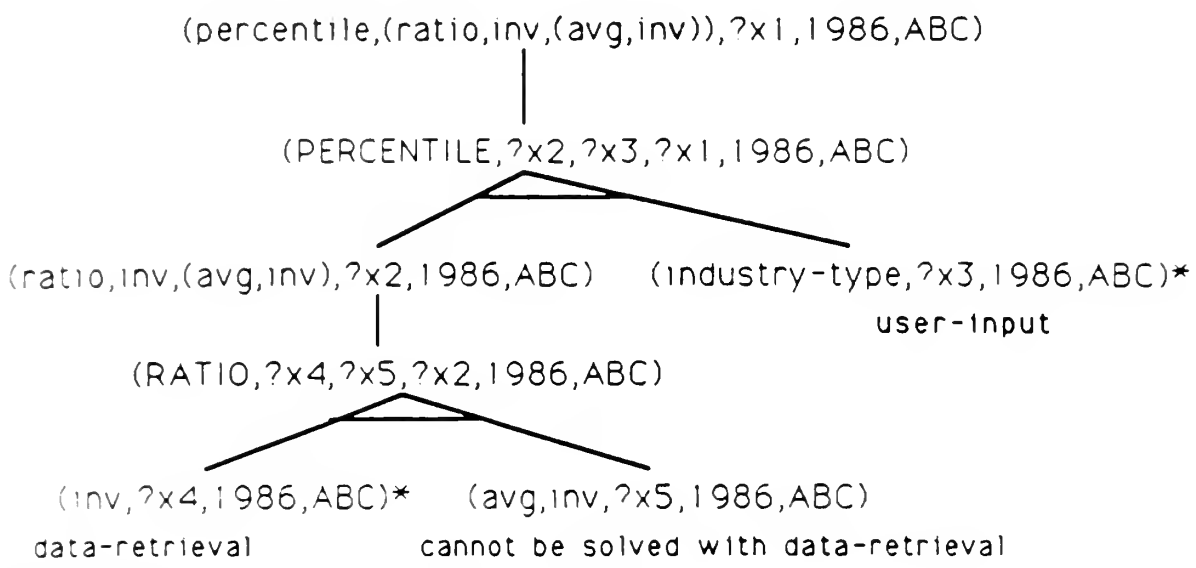


- Since all terminal nodes are unsolvable, this example is treated as a negative example. It modifies the condition of the schema as follows:

G (percentile,(ratio,B/S-var,var2),yr,fn), or
(percentile,(ratio,var1,I/S-var),yr,fn), or
(percentile,(ratio,var1,var2),yr,fn)^(yr<1988)
S (percentile,(ratio,assets,B/S-var),1986,ABC)

Figure A-4. Applying the Schema to a Negative Example

● An example ,(percentile,(ratio,inv,(avg,inv)), ?x1,1986,ABC), with the partial pattern-matching, (inv/var1, 1986/yr,ABC/fn), has the following instantiated solution.



● t then modifies the schema as follows

condition

G (percentile,(ratio,B/S-var,var2),yr,fn), or
(percentile,(ratio,var1,1/S-var),yr,fn),or
(percentile,(ratio,var1,var2),yr,fn)^(yr<1988),or
(percenitle,(ratio,var1,(avg,var2)),yr,fn).

S: (percentile,(ratio,assets,B/S-var),1986,ABC), or
(percentile,(ratio,inv,(avg,inv)),1986,ABC)

solution

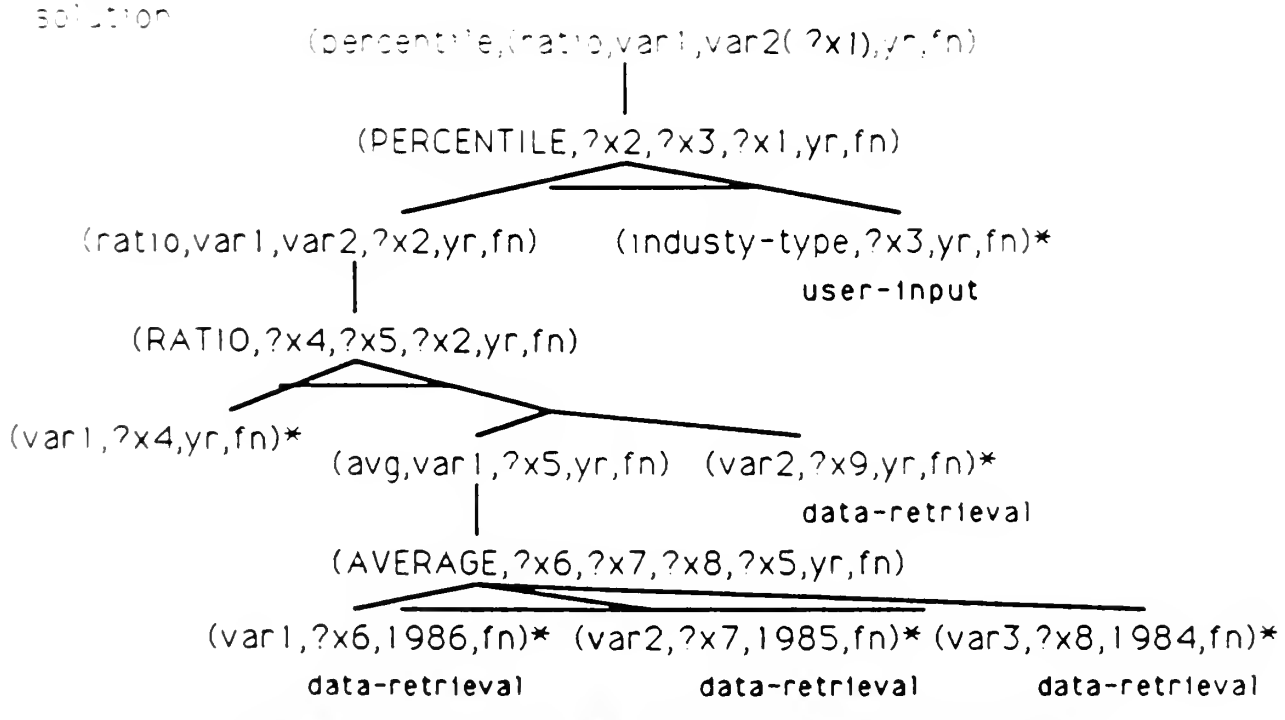


Figure A-5. Applying the Schema to a Near-miss Example